### **PROJECT3: Event-driven Asynchronous Communication**

### **using Mesh Architecture**



|  |  |
| --- | --- |
| **■ 과목** | 인공지능반도체 |
| **■ 담당교수** | 홍상훈 |
| **■ 제출일** | 2022.12.19 |
| **■ 학과** | 전자공학과 |
| **■ 학번** | 2017103780 |
| **■ 이름** | 권용환 |

목차

### GOAL3

### ABSTRACTION3

### BACKGROUND3

### PROPSED ARCHITECUTRE5

### RESULT17

### CONCULUSION19

### REFERENCE20

### APPENDIX20

GOAL

본 프로젝트에서는 여러 개의 Single Core를 Array형식으로 사용하는 프로세서들의 Core 간 Asynchronous Communication 방법을 고안한다.

ABSTRACTION

Single Core를 사용할 때, Core 사이의 communication 방식을 clock-driven으로 할 경우 동시에 여러 코어에서의 communication을 제어하는 것은 쉽지 않다. 정해진 주기마다 동작이 수행되며 충돌을 피하고 순서를 지정하기 위해서는 많은 클럭을 낭비하여 latency가 길어진다. 따라서 이번 프로젝트에서는 Event-driven 방식으로 특정 signal이 들어오면 그에 따른 응답 signal을 주고받아 data를 주고받는 Asynchronous Communication을 고안한다. 각 Core는 독립적으로 동작하며, 시간에 무관하게 동작한다. Mesh Architecture를 이용함으로써 Multicasting이 가능하도록 하여 redundant, parallel, concurrent processing에 강한 multiple access를 위한 system을 구현한다.

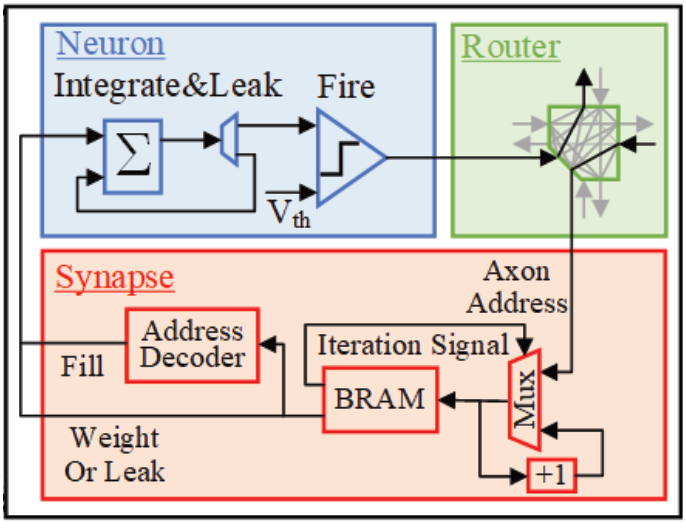
BACKGROUND

Event-driven에 의한 communication을 위한 Click Elements는 [1]에 제시된 것과 같이 여러 형태가 가능하다. [2]에서는 Mesh Architecture를 이용하여 asynchronous communication을 구현한다. [2]의 경우 일반적으로 Event-driven에 사용되는 4-phase가 아닌 2-phase를 이용하였다. 다만, Core의 구조가 지나치게 복잡하고, phase skewing과 같은 추가적인 작업을 요구하고 있다. [3]은 SNN Accelerator를 Event-driven, Mesh Architecture를 이용하여 구현한다. [3]에서는 neuron, synapse에 해당하는 모듈을 소개하며, 실제로 MNIST에 대한 Classification결과까지 제시하고 있다. 그러나, 해당 모듈의 경우 Router의 역할이 매우 간략하게 기술되어 입력과 출력이 어떻게 대응되는지 설명하고 있지 않다. Event-driven의 경우 매우 빠른 속도로 통신이 가능하다는 이점이 있지만 그만큼 complexity가 높기 때문에 논문이나 Article 등 참고할 자료가 많지 않다. 그럼에도 Event-driven 방식의 통신 시스템을 구성하는 대부분의 논문에서는 Core의 Router에서 Local, North, East, South, West의 Channel에서 들어오는 입력을 다시 Local, North, East, South, West로 분배하는 방식을 택하고 있다. 이때, 분배 방식은 크게 두 가지로 나뉜다. 첫번째는, N개의 입력이 들어왔고, M개의 채널이 허용된다면, M개 모두에 입력으로 들어온 데이터를 전달하는 방식이다. 그러나, 이 방식의 경우 입출력의 개수가 다를 수 있고, 출력의 방향을 정하는 알고리즘을 세우기가 까다로운 문제가 있다. 더욱이, 버퍼 등을 이용하여 순차적으로 데이터를 전달하기 때문에 Hardware Cost또한 추가된다. 두번째는, N개의 입력이 들어왔을 때, Multiplexing을 이용하여 하나의 입력을 선택하고, 허용된 모든 채널에 선택된 입력을 전달함으로써 Multicasting을 취하는 방식이다. 이 경우 N개의 입력 중 어떤 기준으로 Selection을 해야 하는지가 중요하다. **PROPOSED ARCHITECTURE**에서는 4-phase에 걸쳐 data를 전달하는 channel과 priority를 rotation하여 5개의 채널에서 들어오는 입력을 선택하는 Allocator를 제안한다. **RESULT**에서는 2x2 Array의 4개의 Core에 random하게 전송되는 data를 바탕으로 4개의 CORE가 clock에 관계없이 불규칙한 distribution을 가지고, 매우 많은 횟수의 통신이 이루어지는 것을 보인다. **CONCULUSION**에서는 본 프로젝트에서 제안하는 Architecture가 갖는 장단점을 분석한다.

PROPOSED ARCHITECTURE

# CORE

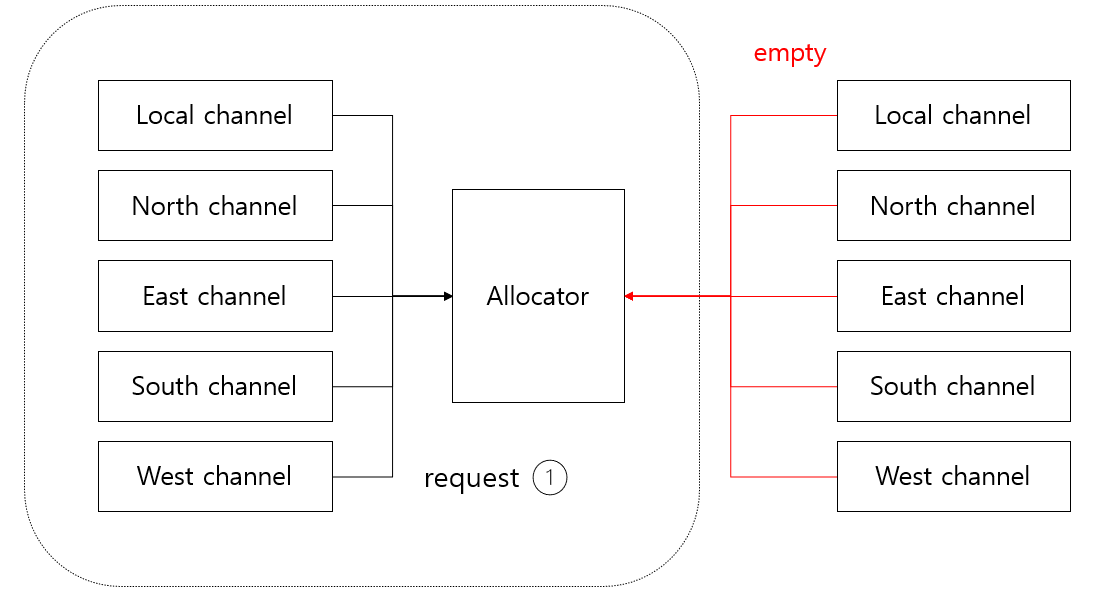
이번 프로젝트는 CORE내부의 neuron은 배제한다. [3]에서 제안하는 CORE의 구조는 Figure 1과 같다.



Figure

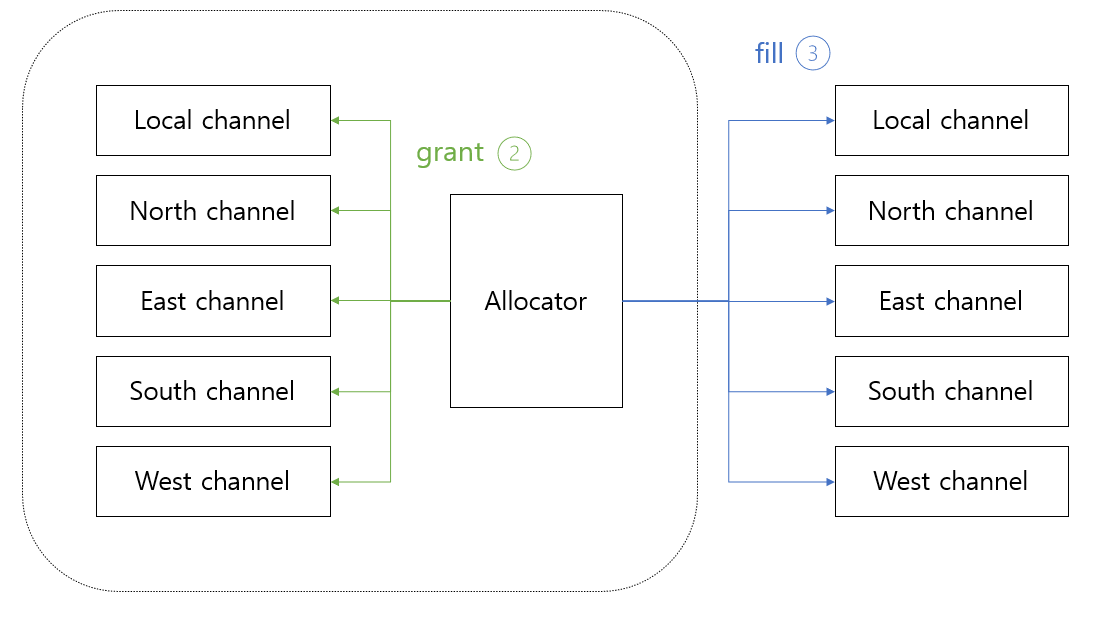
여기서 Neuron, Synapse는 이번에 설계하지 않기 때문에 CORE에는 Router만 필요하게 된다. 우리는 Router의 5개의 Channel 중 Local Channel에 PE(Processing Element)를 삽입하였고, PE는 간단한 1bit shifter가 되도록 하였다.

Figure1을 다시 보면, neuron의 경우 실제 neuron의 개수만큼 instance화하는 것은 아닌 것을 알 수 있다. BRAM에 저장된 data를 synapse에서 꺼내고, Weight와 fill 신호를 neuron 모듈에 전달하여 iteration을 통해 출력을 결정하게 된다. 이는 protocol에 따라 그 형태가 다양해질 수 있음을 의미한다.



Figure

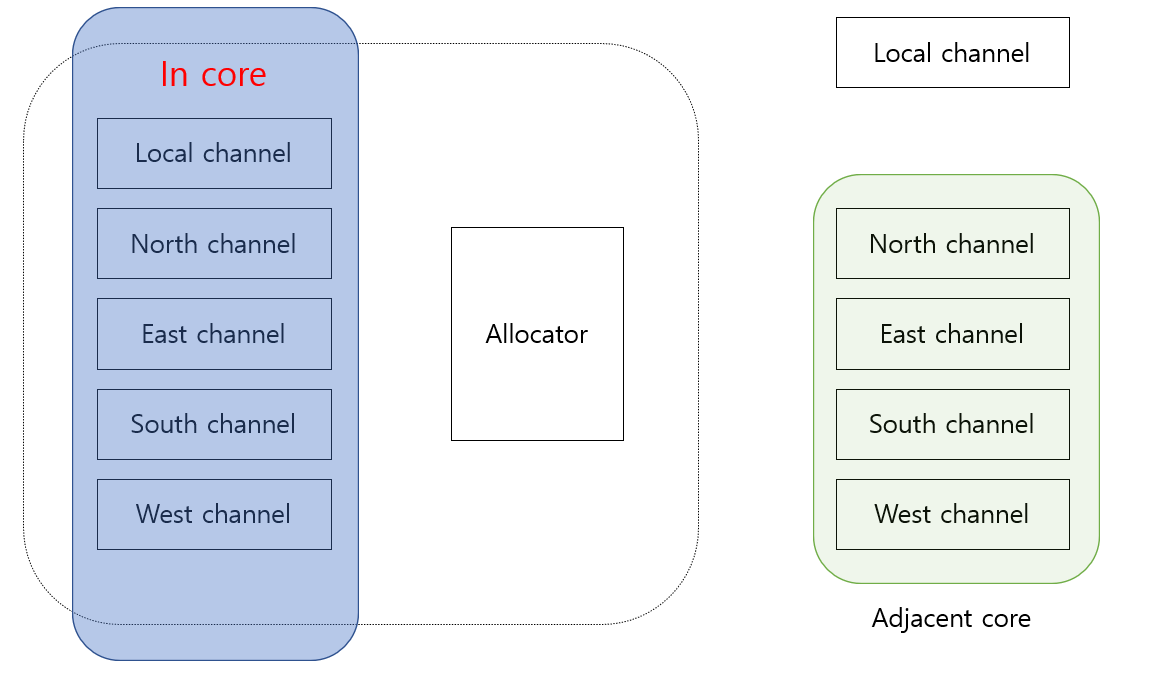
Figure 2는 우리가 설계한 CORE의 구조를 나타낸다. PE는 Local channel에 embedded로 만들어지기 때문에 Figure 1과 비교하여 그 구조는 간단하다. Allocator의 경우 CORE 내부의 channel들로부터 request를 입력으로 받는다. 또한, 근접한 CORE의 channel들로부터 empty를 입력으로 받는다. Figure 2에서 empty 신호를 Allocator로 전송하는 Local channel만 CORE내부의 channel이고, 나머지 North channel, East channel, South channel, West channel은 CORE와 인접하여 붙어있는 CORE들로부터 empty신호를 받는다.



Figure

Allocator는 request신호를 바탕으로 CORE내부에 grant신호를 전달한다. 각각의 channel은 grant신호를 받으면, Allocator로 data를 전송한다. Allocator는 전달받은 data를 trigger로 하여 grant에 따라 Multiplexing한다. 이때의 Mux의 출력을 selected\_data라고 하면, selected\_data를 event trigger로 하여 주변 CORE의 empty신호가 들어오는 모든 channel에 selected\_data를 보내고, fill신호를 pulse로 보내준다.

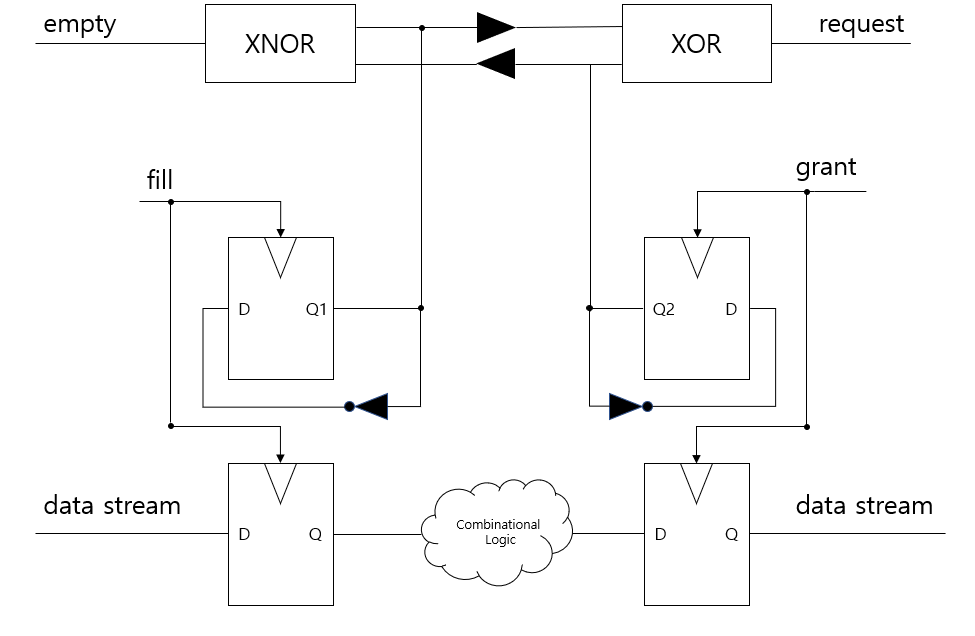
Figure 4는 앞서 언급한 CORE의 내부 channel 영역과 CORE의 외부 channel영역을 보여준다. empty신호는 CORE의 외부 channel에서, request는 CORE의 내부 channel에서 전달받는 것을 기억해야 한다.



Figure

# Channel

Channel은 입력으로 fill, grant신호를 받고, 출력으로 empty와 request신호를 내보낸다. 이때, fill과 grant가 한 번 swing할 때마다, empty와 request는 서로 반대의 signal을 갖게 된다.



Figure

Figure 5의 회로를 보면 Channel의 동작을 더 잘 이해할 수 있다. 초기 상태는 empty는 1, grant는 0인 상태이다. 이 경우 request가 0이기 때문에 grant신호가 swing할 일은 없다. 따라서 다음 동작은 fill신호에 의해서만 정해진다. empty신호를 받은 주변 CORE의 Allocator는 fill신호를 1회 pulse형태를 전송한다. 그렇게 되면, channel은 fill 신호를 받아. Q1의 상태를 transition하게 된다. 그렇게 되면, (Q1, Q2)가 (1,0)이 되기 때문에 empty와 request는 (0,1)로 변하게 된다. 그리고 data stream은 combinational logic으로 전달된다. 그 다음, 현재 CORE의 Allocator는 request를 받아 grant신호를 돌려주게 된다. 이때, grant신호가 들어오면 Q2가 transition하게 된다. (Q1, Q2)는 (1,1)이 되고, empty와 request는 (1,0)으로 돌아가게 된다. 또한, combinational logic의 data stream이 channel의 출력으로 나가게 된다. 정리하면, channel은 4-phase로 동작한다.

**Phase**

phase1. fill 신호를 받아 Q1 transition

phase2. request 신호 송신

phase3. grant 신호를 받아 Q2 transition

phase4. empty 신호 송신

이렇게 4개의 Phase를 iteration하면서 data stream을 전달하는 역할을 하게 된다. [2]에서는 2 phase만에 전송하는 회로를 고안하였으나 중간 단계에서 많은 버퍼를 요구하고, skewing이라는 추가적인 작업이 포함되는 것이 4-phase에 비해 큰 이점이 없다고 판단되어 일반적으로 구성하는 4-phase에 맞게 Channel을 설계하였다.

Table

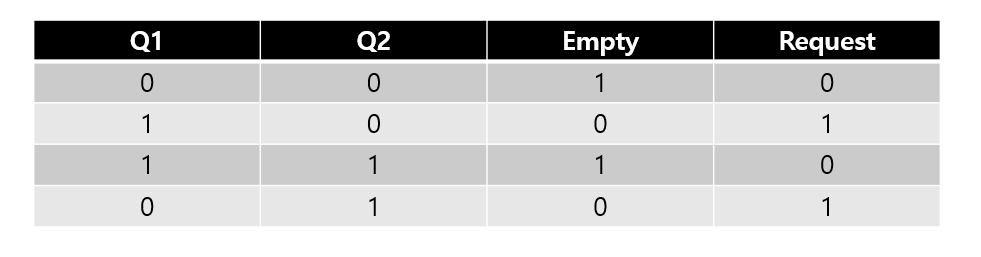
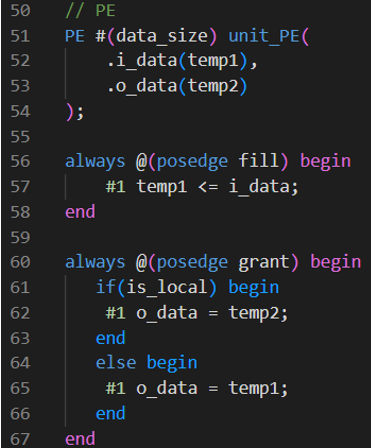


Table 1은 channel의 Q1, Q2에 따른 Empty와 Request를 나타내는 것이다. 또한, Q1과 Q2는 fill과 grant신호에 따라 Table 1의 순서로만 계속해서 iteration하기 때문에, Empty와 Request는 (1,0)과 (0,1)을 반복하는 것을 알 수 있다.



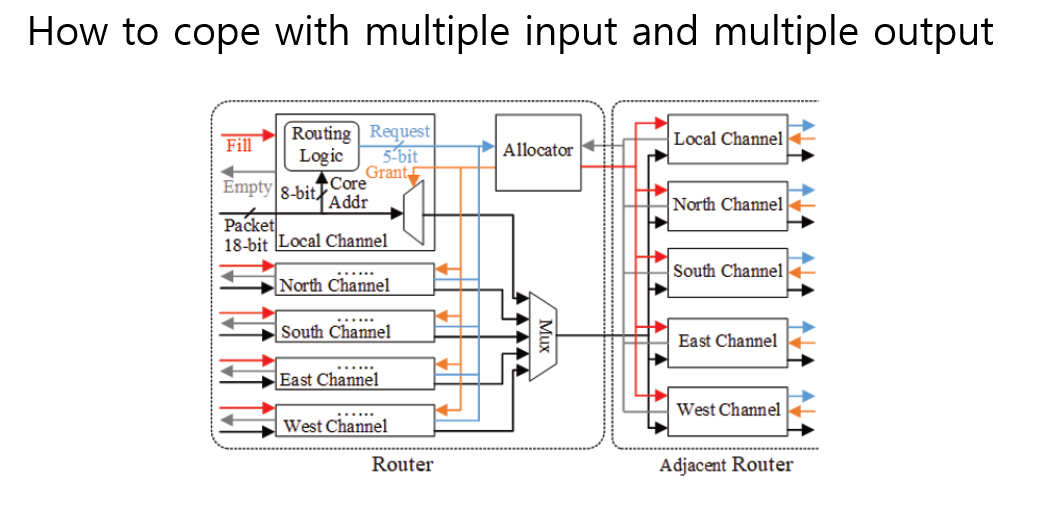
Figure

Figure 6은 channel 내부에 PE가 있음을 보여준다. 60번 라인을 보면 grant신호를 받아 data를 출력하는데, local인 경우와 아닌 경우를 나누고 있다. temp1은 PE의 입력신호인 것을 보면 알 수 있듯 만약, local channel이라면 PE라는 logic을 거쳐서 나오는 data를 출력으로 내보내고, 그렇지 않다면, Q1의 값을 그대로 받아 출력으로 내보내는 것을 알 수 있다.

# Allocator

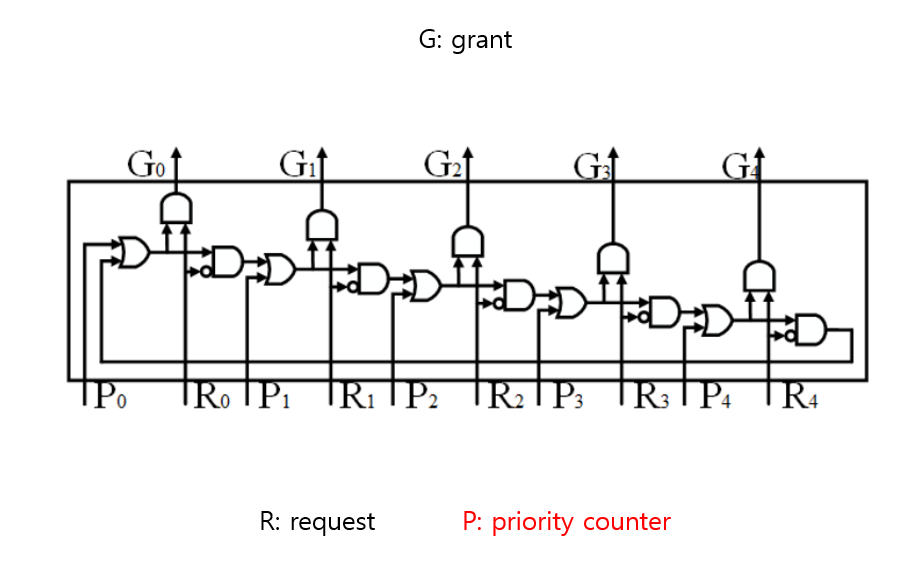
지금까지의 내용을 정리하면 CORE내부에는 5개의 channel과 1개의 allocator가 있다. 그리고, 5개의 channel은 allocator로 request신호를 보내고, allocator는 request에 대한 응답으로 grant신호를 보낸다. grant신호를 받은 channel은 data를 출력하고, allocator는 channel의 출력 중 하나를 선택한다. 선택이 끝나면, 주변 CORE의 channel의 empty신호를 받아 이에 대한 응답으로 fill신호를 내보내고, 선택된 data는 근접한 모든 channel에 일단 전송한다. 이렇게 되면, fill신호를 받은 channel만 그 data를 Q1에 load하게 된다. 이를 통해 주변에 있는 channel들로 multicasting이 가능해진다.

Allocator에서 처리해야 하는 가장 중요한 질문은 Figure7에 잘 나타나 있다.



Figure

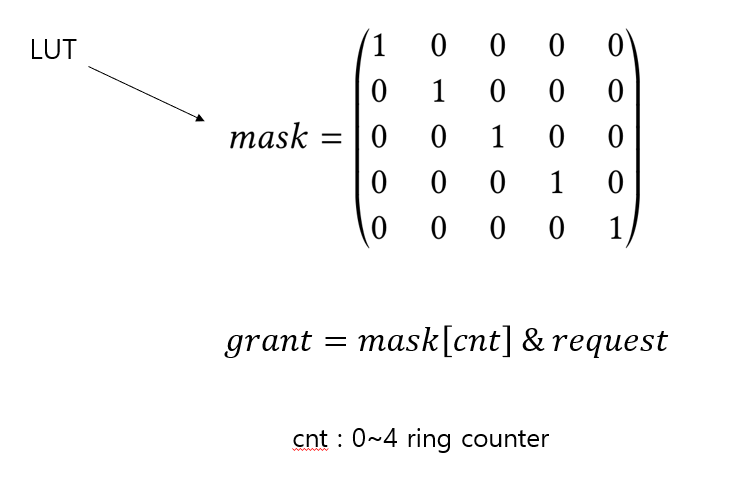
Figure 7은 [3]에서 제안하고 있는 Router구조인데, 하나의 Router는 5개의 channel과 allocator, mux로 이루어져 각각의 channel은 fill, grant신호를 입력으로 받고, empty 신호를 출력으로 한다. 또한, Allocator는 request와 주변 router에서 empty신호를 받고, 내부의 router로 grant신호를 출력하고, 주변 router로 fill신호를 출력한다. 우리는 Figure 7의 구조를 모방하였지만 [3]과는 큰 차이가 있다. [3]에서는 Mux의 기준과 Allocator의 동작을 정확히 밝히지 않고 있다. 우리는 Allocator가 여러 개의 request를 받을 때, 이에 대한 응답을 효과적으로 처리할 수 있는 방법을 찾아 고민했고, Allocator는 일련의 동작 순서가 명확하다. 또한, CORE내부에서는 5개의 channel중 1개만을 선택하지만 선택된 data는 empty를 전송하고 있는 모든 channel로 전달되는 특성에도 차이가 있을 수 있다. 이는 Mux가 N to 1 선택이 아닌 Sorting으로 사용되는 Router들도 적지 않기 때문에 여지를 남겨둔 것으로 보인다.



Figure

Figure 8은 [2]에서 제시된 회로이다. 5개의 channel에서 모두 request가 들어왔을 때, 만약 모든 channel에 grant를 응답으로 보내면 5개의 channel은 Q2의 data를 모두 갱신할 것이다. 이렇게 되면 5개의 channel의 data가 모두 바뀌고 그 중에서 하나만 선택하여 multicasting을 하게 되면, 중간 단계에서 매우 많은 loss가 발생하게 된다. 이 때문에 하나의 channel당 한 번씩 grant를 해줄 필요가 있고, 이를 counter와 결합하여 문제를 해결하려 한다. Figure 8의 경우 P0가 만약 1이라면, R0가 1일 때, G0는 1이 되고, R0가 0일 때, G0는 0이 된다. 즉, P0가 1이면 G0는 R0와 같다. 그러나, P0가 0이라면 G0는 무조건 0을 출력한다. 이에 착안하여 G0, G1, G2, G3, G4 에 각각 priority를 부여할 수 있도록 P0P1P2P3P4가 10000, 01000, 00100, 00010, 00001 을 순환하는 ring counter라면 multiple request에 대하여 priority를 모두 공평하게 분배하면서 중복된 grant에 의해 발생하는 data의 loss를 예방할 수 있다.

Figure 9는 Figure 8의 구조를 implementation할 때, 조금 더 쉬운 방법으로 구성할 수 있도록 one hot encoding 된 5x5 크기의 LUT를 하나 만들고, LUT의 index를 ring counter로 순환하면서 request와 bitwise and 연산을 수행해 grant신호를 만든다.



Figure

그렇게 되면 Figure 9의 한 줄만으로 grant신호를 깔끔하게 출력할 수 있게 된다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Figure

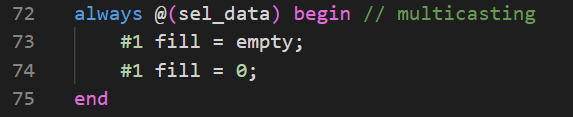
ring counter의 경우 외부에서 변화 속도를 설정할 수 있도록 mask\_cnt\_delay라는 parameter를 입력으로 받도록 구현하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Figure

51라인을 보면 Mux는 grant신호에 반응하지만 channel이 grant신호를 받아 data를 출력하는 delay를 계산하여 selection하도록 하였다. 이를 통해 grant->selection 순서로 동작이 일어남을 알 수 있다.



Figure

그 다음 sel\_data를 event trigger로 하여 fill신호를 empty신호에 응답 후 0으로 보낸다. 여기서 중요한 것은 multicasting이 가능하다는 것과 Allocator가 grant->selection->fill 순서대로 event를 처리한다는 것이다.

Allocator Processing Sequence: grant -> selection -> fill

# Random Pulse Generator

Core1, 2에 random한 fill과 data 신호를 주기 위해서 random pulse generator를 만들었다.

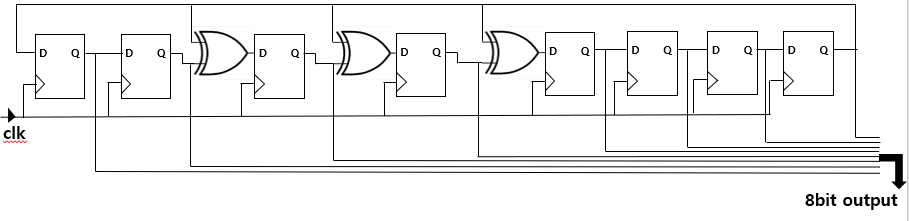


Figure 13

8bit random신호를 위해 Figure 13 같은 구조로 된Linear Feedback Shift Resister(LFSR)을 이용하였다. 8개의 flip-flop을 사용하였고, 2,3,4번째에는 XOR gate를 달아서 feedback이 일어나도록 하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Figure 14

Figure 14의 12~14줄을 보면 구조 그림과 동일하게 2,3,4번째 flip-flop에서 XOR gate를 통해 feedback이 일어나고 있음을 확인할 수 있다. 초기값은 현재 255로 설정되어 있는데, dead end상태인0을 제외한 모든 값으로 설정해도 된다. 코드를 실행하면, 0을 제외한 8bit의 random값 (1~255)이 나오게 된다.

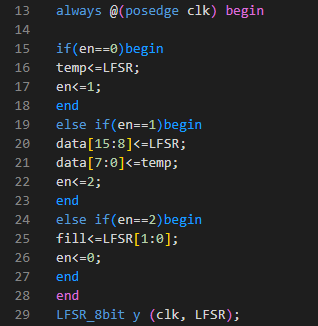


Figure 15

LFSR에서 얻은 8bit의 random한 값을 이용하여 module ‘random pulse generator에서 data와 fill 신호를 얻는다. Figure 15는 Core 1, 2에 각각 하나씩 data와 fill이 필요하므로, 16bit와 2bit의 신호를 만드는 과정이다. en신호는 clk마다 하나씩 증가하는데, en이 0,1 일 때는 8bit random값을 두 번 받아 16bit로 만들어서 data 신호를 얻어내고, en이 2일 때는8bit신호로부터 2bit의 fill 신호를 얻어낸다. 이를 top module에서 반으로 분리하여 사용하게 된다.

RESULT

# 2x2 Core Array

텍스트, 전자기기이(가) 표시된 사진

자동 생성된 설명

Figure 16

Figure 16은 top module의 instance를 나타낸다. 2x2의 Core Array와 Random Pulse Generator를 연결하여 Core1, Core2의 North channel을 통해 fill과 data신호를 전송한다. Core사이의 data의 움직임, fill 신호의 움직임, empty신호의 움직임 등을 simulation을 통해서 볼 것이다. Event-driven이기 때문에 clock-driven과 달리 각 Core사이의 data, fill, empty의 distribution은 매우 불규칙하게 나타날 것이다.

# Simulation Result

Modelsim을 통해 확인한 simulation 결과는 다음과 같다.

텍스트, 전자기기이(가) 표시된 사진

자동 생성된 설명

Figure 17

Figure 17을 보면 가장 하단의 붉은 색이 random data과 random fill신호가 된다. 그 다음 노랑색 신호는 4개 코어 사이의 fill 신호이다. 그 다음, 하늘색 부분은 코어 사이의 empty 신호이고, 그 위의 초록색 영역은 실제 data의 움직임을 보여준다.

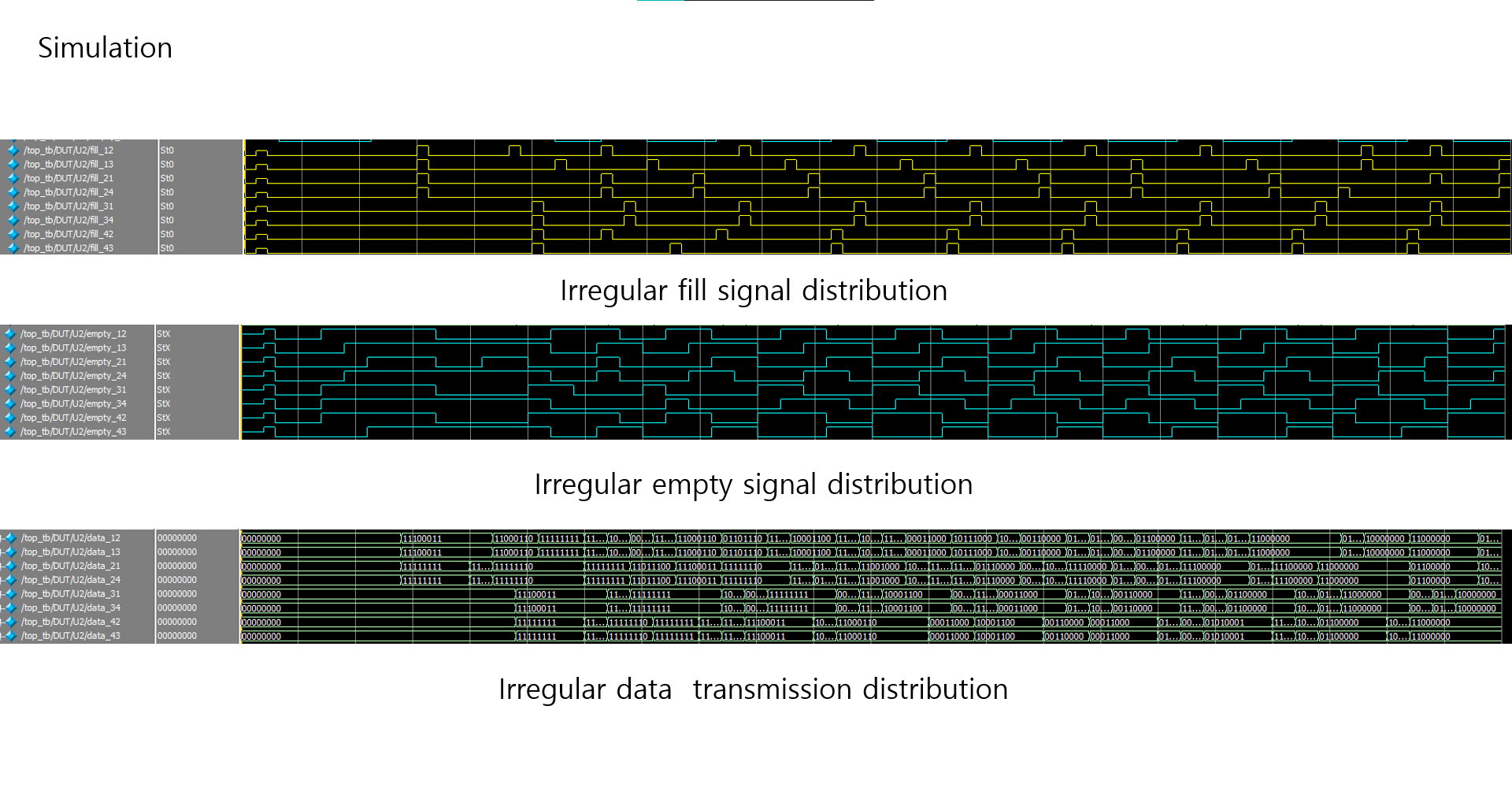


Figure 18

3개의 분포 모두 매우 불규칙하며, 입력으로 들어오는 fill신호에 비해 매우 빠르게 많은 수의 communication이 일어나고 있음을 알 수 있다.

CONCLUSION

이번 프로젝트를 통해 설계한 Multicasting이 되는 Event-driven 방식의 Communication System은 매우 다양한 환경에 사용될 수 있어 보인다. 우선, Resource가 한정적인 Multiple Access를 요구하는 system의 경우 Multicasting을 통해 이를 해결할 수 있다. 또한, Redundancy, Parallelism, Concurrency가 강한 Algorithm의 경우 동시에 여러 코어에 data가 전달되어 연산을 수행할 수 있기 때문에 data의 protocol을 잘 만들어 언제 연산이 끝나는지 기술되면 효과적으로 사용될 수 있을 것으로 기대된다. 또한, channel에서 grant를 priority를 rotate시키면서 하나의 channel을 선택했던 것처럼 empty signal에 대해서도 비슷하게 fill을 줄 수 있다. 예를 들어, empty의 개수를 세서 1개인 경우 해당 channel에 fill을 하고, 2개인 경우 2bit priority counter에 따라서 3개인 경우 3bit priority counter, 4개인 경우 4bit priority counter, 5개인 경우 5bit priority counter를 돌면서 empty에 대한 응답을 줄 수 있다. 개수에 따라 달라지는 이유는 grant의 경우 5개 중 하나만을 선택하면 되기 때문에 event trigger로 ring counter와 request를 사용했지만 fill의 경우 event trigger가 selected data이기 때문에 data의 selection이 일어나면 fill은 0을 출력하면 안 된다. 즉, 어떤 채널은 무조건 선택을 해야 한다. 물론, 앞서 만든 채널이 Click-base architecture인 것처럼 fill을 만드는 것도 예를 들어 fill을 구하기 위한 동작은 Q1, Q2를 XOR한 결과가 1이면 fill을 구하고, 0이면 fill을 구하지 않는 system을 가정하자. 이때, data selection이 일어나면 Q1의 trigger가 되는 신호를 1회 swing시켜 Q1을 1로 만든다. 그러면 temp fill을 구하는 연산을 수행하고, temp fill을 event trigger로 하여 temp fill이 0이 아니면 Q2의 trigger가 되는 신호를 1회 swing시키면서 Q2를 0으로 만들어 temp fill이 구해지는 것을 멈추고, fill에 temp fill을 전달한다. 그러나, 이 방식에서 주의할 것은 temp fill이 수정되는 시간이 Q2의 trigger신호를 swing시켜 Q1과 Q2의 XOR신호가 0으로 바뀌는 시간보다 길어야 한다는 것이다. 그 사이에 temp fill이 수정되어 버리면 fill 신호는 잘못 전달된다.

REFERENCE

[1] Ad Peeters, Frank te Beest, Mark de Wit and Willem Mallon. Click Elements An Implementation Style for Data-Driven Compilation.

[2] Anping He, Guangbo Feng, Jilin Zhang, Pengfei Li, Yong Hei and Hong Chen. Click-Based Asynchronous Mesh Network with Bounded Bundled Data.

[3] Jilin Zhang, Hui Wu, Jinsong Wei, Shaojun Wei and Hong Chen. An Asynchronous Reconfigurable SNN Accelerator With Event-Driven Time Step Update.

APPENDIX

현재까지의 프로젝트는 어떻게 매우 빠르게 CORE간의 통신을 할 수 있는지에 대한 주제였다. 남은 주제는 SNN에서 어떻게 특정 CORE의 neuron들의 도착지에 데이터를 정확히 전송하고 연산이 수행될지를 결정해야 한다. 이를 위해서는 CORE에 있는 뉴런들의 다음 뉴런의 id와 각 뉴런의 연산횟수를 지정하는 과정이 필요하다. 먼저, data stream의 protocol을 지정하고, neuron의 algorithm과 architecture를 만들 것이다. 또한, 이제 neuron을 만들 것이기 때문에 Local channel의 구조도 변경될 것이다.

# PROTOCOL

각각의 neuron에 다음 neuron의 id와 각 neuron의 연산횟수를 부여하는 단계, 연산을 진행하는 단계, 각 neuron에 저장된 정보를 초기화하는 단계로 구성된다.